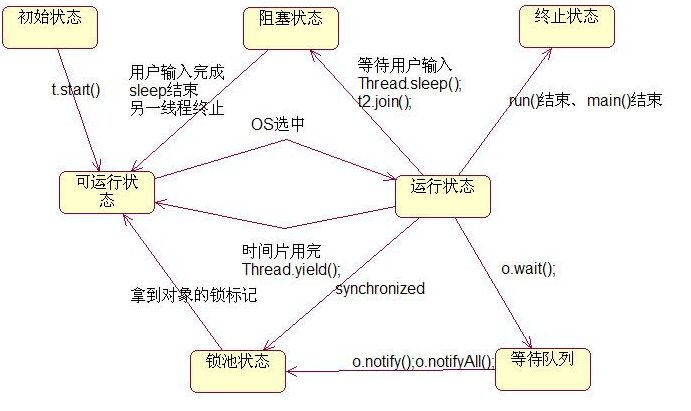
线程的生命周期



1. 线程状态：
   1. New 新建
   2. Runnable 可运行
   3. Running 运行 （调用getState()时显示为Runnable）
   4. Blocked 阻塞
      1. I/O阻塞 （不释放锁）
         1. I/O操作完成解除阻塞，进入Runnable状态
      2. 同步阻塞（不释放锁）
         1. 运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中
   5. Waiting 等待
      1. Object.wait() （释放锁）
         1. 由其他对象通过notify或notifyAll通知等待线程，进入Runnable状态
      2. Thread.join() （获取被调用者的锁）
         1. 调用该方法的线程阻塞，让指定的线程加入运行，直至该线程执行完为止，进入Runnable状态
      3. 等待java.util.concurrent库中的Lock或Condition时
   6. Timed waiting 计时等待（有超时参数，直到超时期满或接收到通知状态解除）
      1. Thread.sleep （不释放锁）
      2. Object.wait （释放锁）
      3. Thread.join （调用者释放锁）
         1. 等待该线程终止的时间最长为 millis 毫秒。超时为 0 意味着要一直等下去
      4. Lock.tryLock （不释放锁）
      5. Condition.await （释放锁）
   7. Terminated 被终止
2. 解除阻塞的CPU 调度策略
   1. 当一个线程被阻塞或等待时或终止时，另一个线程被调度为运行状态；
   2. 当一个线程被重新激活，调度器检查它是否具有比当前运行进程更高的**优先级**，有则从当前运行线程中挑选一个剥夺其运行权（变为Runnable），选择一个新的线程运行
3. 进入运行状态后情况
   1. run()方法或main()方法结束后，线程就进入终止状态；
   2. 当线程调用了自身的sleep()方法或其他线程的join()方法，就会进入阻塞状态（该状态既停止当前线程，但并不释放所占有的资源）。当sleep()结束或join()结束后，该线程进入可运行状态，继续等待OS分配时间片；
   3. 线程调用了yield()方法，意思是放弃当前获得的CPU时间片，回到**可运行状态**，这时与其他进程处于同等竞争状态，OS有可能会接着又让这个进程进入运行状态；
   4. **锁池状态**： 当线程刚进入可运行状态（注意，还没运行），发现将要调用的资源被synchroniza（同步），获取不到锁标记，将会立即进入**锁池状态**，等待获取锁标记（**这时的锁池里也许已经有了其他线程在等待获取锁标记，这时它们处于队列状态，既先到先得 FIFO**），一旦线程获得锁标记后，就转入可运行状态，等待OS分配CPU时间片；
   5. **等待队列**：当线程调用wait()方法后会进入**等待队列**（进入这个状态**会释放所占有的所有资源**，与阻塞状态不同），进入这个状态后，是不能自动唤醒的，必须依靠其他线程调用notify()或notifyAll()方法才能被唤醒（由于notify()只是唤醒一个线程，但我们由不能确定具体唤醒的是哪一个线程，也许我们需要唤醒的线程不能够被唤醒，因此在实际使用时，一般都用notifyAll()方法，唤醒有所线程），**线程被唤醒后会进入锁池**，等待获取锁标记。